High Performance Data Broadcasting: A Comprehensive Systems' Perspective

Peter Triantafillou, R. Harpantidou, M. Paterakis

Dept. of Computer Engineering
Technical University of Crete

peter@ced.tuc.gr
roula@telecom.tuc.gr
pateraki@telecom.tuc.gr

Abstract. Broadcast scheduling algorithms have received a lot of attention recently, since they are important for supporting mobile/ubiquitous computing. However, a comprehensive system's perspective towards the development of high performance broadcast servers is very much lacking. With this paper we attempt to fill this gap. We contribute four novel scheduling algorithms that ensure the proper interplay between broadcast and disk scheduling in order to attain high performance. We study comprehensively the performance of the broadcast server, as it consists of the broadcast scheduling and the disk scheduling, algorithms. Our results show that the contributed algorithms outperform the algorithms, which currently define the state of the art. Furthermore, one of our algorithms is shown to enjoy considerably higher performance, under all values of the problem and system parameters (such as the skew of access distributions, the system load, the data object sizes, cache- and disk-intensive workloads, etc.). An important conclusion of this study is that broadcast scheduling algorithms have only a small effect on the overall broadcast system performance, a fact that necessitates the refocusing of related research.

1. Introduction

Mobile computing and wireless networks are quickly evolving technologies that are making ubiquitous computing a reality. As the population of portable wireless computers increases, mechanisms for the efficient transmission of information to such wireless clients are of significant interest. Such mechanisms could be used by a satellite or a base station to disseminate information of common interest. Many emerging applications involve the dissemination of data to large populations of clients. Examples of such dissemination-oriented applications include information dispersal systems for volatile time-sensitive information such as stock prices and weather conditions, news distribution systems, traffic information systems, electronic newsletters, software distribution, hospital information systems, public safety applications, and entertainment delivery.

Many of the dissemination-oriented applications we mention above, have data access characteristics that differ significantly from the traditional notion of client-server applications as embodied in navigational web browsing technology. A fairly limited amount of data is distributed from a small number of sources to a huge client population (potentially many millions) that have overlapping interests, meaning that any particular data item is likely to be distributed to many clients. Data broadcasting is considered to be an efficient way, in terms of bandwidth and energy, for the distribution of such information for both wireless and wired communication environments and has been extensively studied (see [11], [12], [14] and [15]). Furthermore, broadcast transmission compared to traditional unicast can be much more efficient for disseminating information, because unicast by having to transmit every data item, often identical, at least once for every client who requests it, creates scalability problems as the client population increases. Thus, the main advantage of broadcast delivery is its scalability: it is independent of the number of clients the system is serving. Much of the communication technology that has enabled large-scale dissemination supports broadcast, and in some cases, is primarily intended for broadcast use. For instance, direct broadcast satellite providers, and cable television companies (through the use of high-bandwidth cable modems) are now, or will soon be, capable of supporting multi-megabit per second data broadcast. Intel has also been broadcasting data along with normal TV signals, [9].

1.1 Related Work

The problem of determining an efficient broadcast schedule for information distribution systems has been extensively studied in the past ([1], [2], [3], [4], [5], [10], [11], [12], and [14]). In [1] the authors propose the RxW scheduling algorithm which calculates the product of the number of outstanding Requests (R), times the Wait time (W) of the oldest outstanding request for all data items corresponding to the requests pending in the broadcast server queue. The data item with the highest product value is chosen for broadcast. Therefore, RxW broadcasts a data item either because it is very popular (high R value) or because it has at least one long-outstanding request. It provides a balanced treatment of requests for both popular (hot) and not-so-popular (cold) items.

In [2], Acharya and Muthukrishman, study the scheduling problem arising in on-demand environments for applications with data requests of varying sizes and they introduce an alternative to the response time of a requests metric- the stretch of a request, which seems better suited to variable-sized data items. They present an algorithm called MAX based on the criteria of optimizing the worst case stretch of individual requests. In [3] and [16] memory is assumed available at each user. The management of this memory was considered in order to reduce the mismatch between the push-based broadcast schedule and the user's access pattern. In [4], the authors consider the problem of scheduling the data broadcast such that the access latency experienced by the users is minimized. Push-based and pull-based systems are considered. In [5], algorithms for determining broadcast schedules in asymmetric environments that minimize the wait time are considered. Variations of those algorithms for environments subject to errors, and systems where different clients may listen to different number of broadcast channels are also considered.

1.2 Problem Formulation and System Model

The abundance of dissemination-based applications caused the rapid development of scheduling algorithms for data broadcast. All such algorithms attempt to select which item to broadcast in order to improve performance. The root implicit or explicit assumption of existing scheduling algorithms is that the data items are immediately available in the broadcast servers' main memory, (see [1], [2], [4], [5], [10], and the references therein). This assumption ignores the fact that in many cases data items must be retrieved from secondary storage before they can be broadcasted. They also typically ignore the existence of the broadcast server's cache and the related cache management issues. By ignoring these issues, such scheduling algorithms when used in real systems can cause significant degradation of the broadcast efficiency, or at the very least the reported results, regarding the efficiency of proposed broadcast scheduling algorithms are misleading. In our paper, we take into account the fact that broadcast scheduling, disk scheduling, and cache management algorithms affect the performance of each other and the overall performance of the broadcast server.

With this paper we put forward a comprehensive study from a systems' viewpoint of the problem of pull-based broadcast scheduling. We consider a broadcast server with the architecture shown in Figure 1. All newly generated client requests enter into the broadcast server queue. The requests may need service from the disk server or alternatively, when cache memory exists, the data items may be found in the cache (i.e., they had been retrieved earlier from disk) and they are forwarded directly to the transmitters' queue. From the transmitters' queue all data items are transmitted through the communication channel, reaching the clients that had made the corresponding requests. When a data item is broadcasted, all requests for the particular data item are satisfied simultaneously regardless of the time of their arrival.

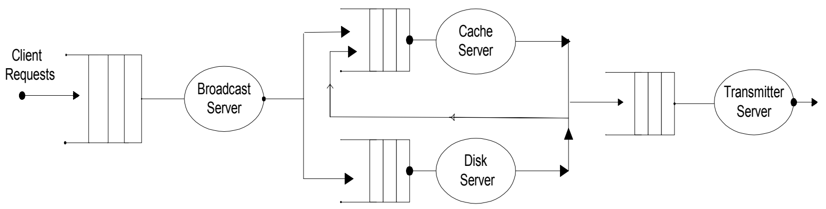


Fig. 1. Broadcast Server Architecture

The system we study consists of a large and possibly time varying client population that requests data items from an information source equipped with a data broadcasting capability. Clients use two independent networks for communicating with the server: an uplink channel for sending requests to the server, and a "listen only" downlink channel for receiving data from the server. When a client needs a data item (e.g., a database object) that cannot be found locally, it sends a request for the item to the server. Client requests are queued up (if necessary) at the broadcast server upon arrival. Requests that correspond to the same item are grouped together forming a multi-request. In the remainder of the paper, we refer to such multi-requests as requests.

We make the following assumption. First, we assume, for simplicity reasons only, that data items are of fixed-length (e.g., database objects). Second, we assume that clients continuously monitor the broadcast channel after they send a request to the server and we do not consider the effects of transmission errors, so that all clients waiting for a data item receive it when is broadcasted by the server. We ignore the delay for sending requests via the client-to-server uplink, which we expect to be small compared to the latency of obtaining broadcast items from moderately or heavily loaded servers.

1.3 Overview of Contributions

Our study is comprehensive in that it considers the interplay between the broadcast scheduling algorithm and the disk scheduling algorith. The contributions are:

- We propose mechanisms that ensure the required interplay of the above algorithms in order to ensure high performance. These mechanisms consist of four novel scheduling algorithms.
- We show that without such mechanisms the algorithms for broadcast scheduling found in the literature can be of little practical use.
- We conduct a detailed performance study: we quantify the expected performance under different values of the problem parameters and we identify the critical mechanisms that limit performance under different configurations.

1.4 The Remainder of the Paper

The remainder of the paper is organized as follows. In Section 2, we describe the scheduling algorithms ADoRe, FLUSH, OWeiST, RxW/S that we propose. In Section 3, we present the simulation model, the performance metrics and the performance behavior of these algorithms. In Section 4, we introduce a novel cache management mechanism and we present performance results. In Section 5, we include the transmitter and we present performance results for the FLUSH algorithm. Finally, we conclude this paper in Section 6.

2. Broadcast and Disk Scheduling Algorithms

The broadcast scheduling algorithm that we have chosen is the exhaustive RxW algorithm, which appears to be a practical, low-overhead scalable scheme that requires no a-priori advanced knowledge (such as the access probabilities of items), [1]. Our group has performed performance studies comparing RxW with other algorithms (e.g., the algorithms in [2], and [5]), and we have found it to have the best performance. These are the reasons we have chosen it as broadcast scheduling algorithm.

For the disk scheduling we selected the C-LOOK algorithm [18], unless stated otherwise. The C-LOOK algorithm sorts data items to be retrieved from the disk in ascending order of their cylinder position on the disk. The read-write head is only moved as far as the last request in each direction. It services requests from the service queue as it reaches each cylinder, until there are no requests in the current direction. Then, it immediately returns to the first requested item of the other end, without servicing any requests on the return trip, and repeats the process.

In this section, we disregard the existence of a cache and we assume that each requested data item must be retrieved from the secondary storage. This is done for two reasons: first in order to measure the impact of the disk system in the performance of the server, and second, because in applications where the data items will be very large and the distribution of the requests to data items will not be skewed, the cache will have little impact. Furthermore, in some system configurations the cache size might be quite small (e.g., as an extreme example, consider a broadcast server on a network attached disk which has a cache size that is a negligible percentage of the database size).

2.1 Combining Separate Broadcast and Disk Scheduling Algorithms

The first of the algorithms we present below extend and combine the RxW broadcast scheduling algorithm with a disk scheduling algorithm through various mechanisms. In the literature ([1]), RxW is described as being applied after the transmission of the previously selected item from the broadcast queue has been completed. The second uses FCFS, as broadcast scheduling algorithm. The algorithms depend on the C-LOOK disk scheduling algorithm.

2.1.1 The ADoRe Algorithm: Active Disk on Requests

The ADoRe is a fairly simple algorithm. When the disk becomes idle, K or fewer requests (corresponding to the case when the broadcast queue does not contain K requests) are directed from the broadcast server queue to the disk scheduler queue to be served. If there are more than K outstanding requests in the broadcast server queue, the RxW algorithm is applied and a group of K requests with the highest RxW values are directed to the disk queue.

A straightforward implementation of the RxW algorithm would imply that each time it is called the broadcast scheduling algorithm forwards a single request to the disk system and once the item is retrieved and broadcasted, RxW is called again to pick the next item, and so on. This, obviously, results in poor disk system performance. The ADoRe algorithm attempts to avoid this shortcoming by using RxW to select a group of K requests. The parameter K allows the formation of groups of requests, in contrast to the straightforward implementation of the RxW algorithm mentioned before. Intuitively, the ADoRe algorithm tries to keep the disk system as highly utilized as possible, while on the other hand it tries to reduce the average disk service time by forwarding a group of requests to be served on which the disk scheduling algorithm seek optimization will produce better results. For example, a C-LOOK sweep with 10 requests served (i.e., when K=10) will take less time than running the RxW algorithm 10 times picking one request at a time and giving it to the disk. Therefore, if K equals to 1 the ADoRe algorithm resembles the straightforward implementation of the RxW algorithm since it directs one request from the broadcast server queue by applying the RxW algorithm, to the disk server queue every time the broadcasting of the previously selected data item is completed.

2.1.2 The FLUSH Algorithm

The requests in the disk queue are served using the C-LOOK algorithm. The FLUSH algorithm manipulates differently the requests on the broadcast server. Every time the disk finishes the service of a single request, all the requests in the broadcast server queue are flushed to the disk server and incorporated into the C-LOOK lists.

2.2 Amalgamating Broadcast and Disk Scheduling Algorithms

The amalgamated algorithms combine information available at the broadcast server and at the disk server.

2.2.1 The OWeiST Algorithm: Optimal Weighted Service Time

The OWeiST algorithm attempts to improve performance in two ways. First, it exploits information available at the broadcast server and at the disk server. Second, it employs a different disk scheduling algorithm, which for larger groups of requests, introduces further optimizations.

According to the OWeiST algorithm whenever the disk becomes idle, K or fewer requests as in the AdoRe algorithm, are being selected from the broadcast queue and forwarded to the disk queue. The service of the requests is being carried out in such order that the sum of the products R times Disk Service Time of the requests is kept minimum. The operation of the algorithm is as follows. We maintain a graph of K requests. The edge connecting two requests r_i and r_j has a label $R_j x S_j$ where R_j is the number of requests in the broadcast queue for item j and S_j represents the disk access cost to access item j, given that the previously retrieved item from the disk was item i. The disk access cost contains both the seek time from the cylinder of item i to the cylinder of item j, plus the rotational delay necessary to access item j once the disk head is positioned on j's cylinder, plus the time to retrieve item j from the disk. The algorithm computes all possible permutations for the K requests and selects the optimal permutation that gives the smallest total weighted cost. Note that, obviously, this is analogous to computing a solution to the Traveling Salesman Problem (TSP). However, by bounding the value of the parameter K we can control the overhead involved in computing the optimal service schedule. Notice that, when K equals 1 OWeiST is identical to ADoRe with K equal to 1.

2.2.2 The RxW/S algorithm

We also propose the amalgamated algorithm RxW/S, where S is the disk service time, which takes into account information of the broadcast scheduling algorithm (i.e., R and W for every requested data item) and the disk service overhead. It is an one step algorithm, which is being activated whenever the disk becomes idle and selects a request from the broadcast queue to be serviced from the disk. The selected request, is the one that has the higher value of the fraction (RxW)/S. In this algorithm there is no grouping of requests because requests are directed to the disk one at a time. The selection of the data item to be broadcasted, favors items of high RxW value and low disk access times.

3. Performance Study

The performance of the algorithms has been studied through simulation. The simulations were executed on a Pentium II PC, 400MHz. Each run simulates the transmission of one million data items. We observed, that by simulating 1,000,000 transmitted data items, we were able to estimate with accuracy the steady-state algorithms performance.

Table 1 shows the parameter setting for the simulated disk system. We assume that the database consists of 10,000 16KB (or alternatively 200KB data items).

Disk Characteristics	
Cylinders	6,900
Surfaces	12
Sector Size	512
Revolution Speed	10,000 RPM
Number of Zones	20
Average Transfer Rate	12 MBps

Table 1.

3.1 Simulation Model

We developed a model, as depicted in Figure 1. We used a Request Generator, which generates a stream of requests according to a Poisson arrival process. The request arrival rates we used in our simulations vary between 10 and 500 requests per second.

Requests are generated from the Request Generator and then they enter the broadcast server queue. By the application of the algorithms in section 3, requests are directed to the disk server queue where the corresponding data items are retrieved from disk and are then forwarded to the transmitters' queue (see paragraph 3.3), or in the presence of cache memory the data items that are located in the cache are sent directly to the transmitters' queue (see paragraph 4.2). When a cache exists, all the data that are forwarded to the transmitters' queue are first moved from disk to the cache (provided that they are not already located in the cache). The transmitter conveys all the data items to the clients through the channel link. Finally, the statistics collector records all relevant statistics in order to measure the performance.

In our simulation, it is assumed that the request probabilities of all data items follow a Zipf distribution. The Zipf distribution may be expressed as follows:

$$p_i = c\,(1/i)^{\theta}, \quad 1 \le i \le M \qquad \textbf{(1)}$$

where $c = 1/\sum_{i=1}^{M} (1/i)^{\theta}$ is a normalizing factor, and θ is a parameter referred to as the access skew coefficient. The distribution becomes increasingly "skewed" as θ increases, [5], [6]. We will report results for two values for θ; θ = 0 (uniform distribution), and θ=1.17 (highly skewed access distribution).

3.2 Performance Metrics

In client-server information systems, the user response time, namely the time between the arrival of the request at the broadcast server and its service, is one of the most important factors for evaluating the systems' performance. A metric that gives an overall view of the response time of all clients in the system is the mean response time. As is remarked in [6], it is natural in the real world some users' demand patterns to completely differ from the overall demand pattern and their own response time may be much worse than the overall mean. In this paper, we address this problem by adopting as a performance metric an *index of fairness* that always lies between 0 and 1, [7]. We have chosen this metric over the square coefficient of variation since the fairness index is a further normalization giving a number between 0 and 1. This boundedness aids intuitive understanding of the fairness index. For example, an algorithm with a fairness index of 0.10 means that it is unfair to 90% of the users, and an algorithm with a fairness index of 0.90 means that is fair to 90% of the users. The fairness index, if *n* contending users are in the system such that the response time of the request of the ith user is denoted by x_i, is defined as follows:

$$f(x) = \frac{[E[x]]^2}{E[x^2]} = \frac{\left[\sum_{i=1}^{n} x_i\right]^2}{n\sum_{i=1}^{n} x_i^2} \quad , \quad x_i \geq 0 \qquad \textbf{(2)}$$

where x is the random variable denoting the response time of a client's request. We have also considered the ratio of the standard deviation to the mean response time of a request, (i.e., the square root of the square coefficient of variation), as an additional fairness indicator.

3.3 Performance Results

The results we present below are only a small indicative sample of the results we have obtained since we cannot present them all, due to the space limitations. We conducted a number of experiments under different combinations of the arrival rate, the grouping parameter K (for algorithms ADoRe and OWeiST), and the access skew coefficient θ. The two primary performance metrics, the mean response time and the fairness index, are plotted versus λ, for different values of the parameter K. The CPU overhead for the RxW algorithm and for calculating the optimal permutation of the OWeiST algorithm is not included in the mean response time results[1]

Figures 2 through 5, present the results under the assumptions of no cache memory available and of infinite broadcast channel speed. The response time of a

[1] The CPU time (overhead) of the application of the RxW algorithm on the broadcast server queue was estimated to be less than 1ms and the corresponding time for the disk scheduling mechanism of the OweiST algorithm for figuring out the optimal permutations when K equals 5 was estimated to be approximately 1ms. These estimations were based on experiments executed a Pentium II PC, 400MHz.

client's request corresponds to the time between the arrival of the request at the server and the retrieval of the corresponding data item from the disk.

Figures 2 through 4 present the mean response time of the proposed scheduling algorithms, ADoRe, FLUSH, OWeiST, and RxW/S. As we mentioned in section 2, the ADoRe algorithm with K equal to 1 resembles the RxW algorithm. Figure 2, presents the mean response time (in milliseconds) versus the arrival rate λ, for θ=1.17, 16KB data item size, and K=1 (which means that each group of the requests contains 1 element). We observe that as the aggregate request arrival rate increases beyond 50 (λ>50), the mean response time of the ADoRe, and the OWeiST is sharply increased, reaching the value of 590ms for arrival rate λ=120 requests/sec. On the contrary, the mean response time is maintained low for the FLUSH, and the RxW/S algorithms, with mean response time values less than 130ms for the RxW/S and less than 100ms for the FLUSH, with arrival rate λ=120 requests/sec. This demonstrates that the latter two algorithms perform considerably better. The same trend is observed for 200KB data item size, as is shown in Figure 3. The difference in the mean response time is due to the retrieval time from the disk of the 200KB data item, which is a multiple of the corresponding time of the 16KB data items.

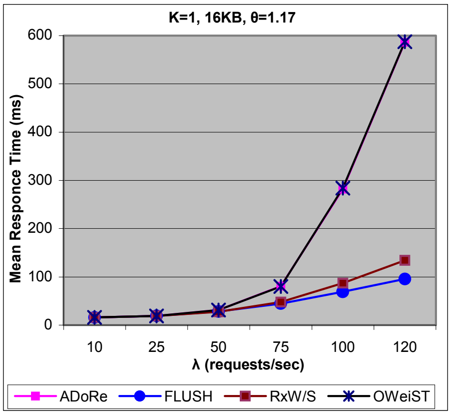


Fig. 2.

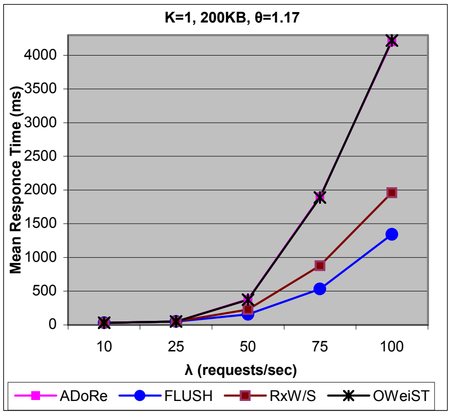
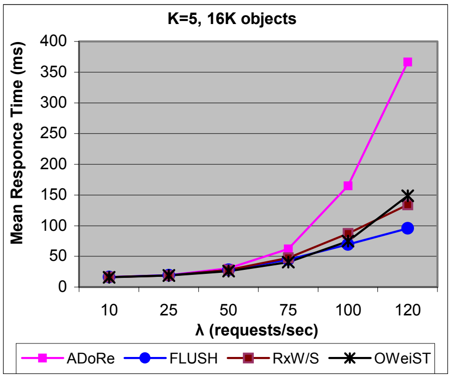


Fig. 3.

In Figure 4 the value of K increases to 5 and we notice similar behavior. Notice that K refers only to ADoRe and OWeiST algorithms, while the curves of FLUSH and RxW/S are the same as in Figure 2. OWeiST performs much better than before, and the mean response time is similar to that of the FLUSH and the RxW/S for all λ values examined. This improvement of the OWeiST was expected, since its optimization (i.e., the calculation of the shortest path for visiting all K data items on the disk in accordance to their popularity) introduces greater benefits as K increases. ADoRe with K equal to 5 performs slightly better than with K equal to 1, because the RxW algorithm forwards a group of requests to the disk and not just one at a time.

The improved performance of FLUSH over the other three algorithms, as λ increases, shown in Figures 2 through 4 was expected since FLUSH forwards requests to the disk server as they arrive, increasing the number of requests that are waiting to be serviced in the disk queue. This gives the disk scheduling algorithm (C-LOOK) the chance to further optimize the disk access time. The OweiST algorithm cannot do that since given the NP-Completeness of calculating the optimal schedule, the disk system queue must be relatively small. The critical observation is that the broadcast server must keep the disk server as busy as possible and this outweighs in importance any improvement from the other disk scheduling algorithms.

Figure 5 shows the fairness index plotted versus the request arrival rate λ for the case of 16KB data item size, and K equal to 1. We observe that FLUSH and RxW/S have a fairness index above 70% even for increased arrival rates, while the fairness indices of ADoRe and OWeiST drop abruptly. Even for 200KB data item sizes (due to lack of space these results are not shown), FLUSH maintains a fairness index around 70% when the fairness indices of the other algorithms drop well below 50%.



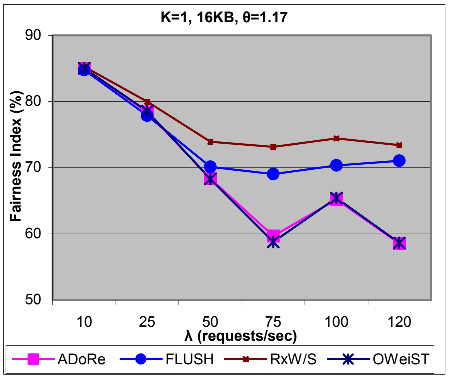


Fig. 4.

Fig. 5.

As an additional fairness metric we have examined the ratio of the standard deviation to the mean response time. FLUSH maintains the value of this ratio below 0.7 for all values of the arrival rate, and data item sizes we examined. We also simulated the demand for data items using the Uniform distribution (θ=0). As expected, the mean response time of all algorithms for θ=0, increases rapidly with arrival rate increases (compared to the results in Figures 2, 3, and 4). This is due to the decreasing possibility of serving more than one client by a single broadcast. FLUSH, however, continues to perform best and has a fairness index above 65%.

4. Contributions and Concluding Remarks

Looking at related work for broadcasting scheduling, one can find several interesting algorithms for deciding which data item to pick for broadcasting. However, all these algorithms make the (implicit or explicit) assumption that the chosen data item is immediately available to the transmitter for broadcasting. In a real system this obviously does not hold. This fact begs the question of how all the basic system components of a broadcast server's system infrastructure should interact in order to build high performance broadcast servers. With this paper we attempt to address this question. We put forward a comprehensive study from a system's viewpoint of the problem of broadcast scheduling. Our study is comprehensive in that it considers the interplay between the broadcast scheduling algorithms, the disk scheduling algorithms, and the cache management algorithms.

We study the interplay between broadcast and disk scheduling algorithms, which will be the critical performance issue in applications and system configurations where the impact of caches will be secondary. We propose four novel scheduling algorithms, the ADoRe, FLUSH, OWeiST, and RxW/S classified under two categories: those that combine separate broadcast and disk scheduling algorithms and those that amalgamate the information available at the broadcast queue and at the disk queue, producing a single scheduling criterion. We study their performance in terms of mean response time and their fairness under different values of the problem parameters (system load, access distributions, object sizes, etc.).

The major conclusions of this work are:

In environments where the broadcast server depends heavily on the disk system, (i.e., it is disk-intensive as opposed to cache-intensive) the critical issue is to design mechanisms which ensure that the server keeps the disk system highly utilized and allowing the disk scheduling algorithm to perform its optimizations. Our results show that our FLUSH algorithm can outperform significantly other mechanisms, which are based on either more efficient disk scheduling algorithms or on algorithms based on combining information at the broadcast and the disk queues.

Our performance study has shown that the proposed algorithms namely, ADoRE, FLUSH, OWeiST, and RxW/S can achieve significantly better performance than that of the RxW algorithm (as it is straightforwardly implemented using the ADoRe (K=1) mechanism) under a large variation of the values of the basic problem parameters (e.g., access skew distribution, system load, object sizes, cache or disk intensive workloads etc.). Furthermore, our results show that FLUSH, consistently outperforms all others under all the above workload and system parameters, while being fair.

A conclusion worthy of special notice is that we have actually found that "efficient" broadcast scheduling algorithms found in the literature have a negligible impact. Specifically, if instead of employing the RxW algorithm at the broadcast queue we employed FCFS scheduling the difference in the overall performance of the broadcast system would barely be noticeable. Notice that, in the FLUSH algorithm, every time we have an arrival, it is passed to the disk system. Viewed differently, this implies that FLUSH essentially uses a FCFS scheduling at the broadcast queue. This conclusion is a strong indication that the main attention of most related research efforts needs to be refocused.

References

[1] D. Aksoy, M. Franklin, "Scheduling for Large-Scale On-Demand Data Broadcasting", Proc. IEEE InfoCom Conf., San Francisco, CA, 1998.

[2] S. Acharya, S. Muthukrishnan, "Scheduling On-demand Broadcasts New Metrics and Algortihms", Proc. ACM/IEEE Int. Conf. MobiCom '98, Dallas,October 1998.

[3] C. J. Su, and L. Tassiulas, "Joint Broadcast Scheduling and User's Cache Management for Efficient Information Delivery", Int. Conf. MobiCom '98, Dallas,October 1998.

[4] C.-J. Su and L. Tassiulas, “Broadcast Scheduling for Information Distribution”, Proc. IEEE INFOCOM Conf.,CA, 1997.

[5] N. H. Vaidya and S. Hammed, “Data Broadcast in Asymmetric Wireless Environment”, Proc. of Workshop on Satellite-based Information (WOSBIS), New York, November 1996.

[6] S. Jiang and N.H. Vaidya, “Response Time in Data Broadcast Systems: Mean, Variance and Trade-Off ”, Proc. of Workshop on Satellite-based Information (WOSBIS), Texas, October 1998.

[7] R. Jain, D-M. Chiu, W.R Hawe, ”A Quantitative Measure of Fairness and Discrimination for Resource allocation in Shared Computer Systems”, DEC-TR-301, September 1984.

[8] E.J. O’Neil, P.E. O’ Neil and G. Weikum, ”An Optimality Proof of the LRU-K Page Replacement Algorithm”, Journal of the ACM, Vol. 46, January 1999.

[9] Intel Corporation, Intel Intercast Technology, http://www.intercast.com, 1997.

[10] S. Acharya, M. Franklin, S. Zdonik, “Balancing Push and Pull for Data Broadcast”, Proc. ACM SIGMOD Conf., Tuscon, Arizona, May 1997.

[11] K. Stathatos, N. Rousopoulos, and J.S. Baras, “Adaptive Data Broadcast in Hybric Networks”, in Proc. VLDB, 1997.

[12] H. Dykeman, M. H. Ammar, and J. Wong, ”Scheduling algorithms for videotext systems under broadcast delivery”, in Proc. International Conference of Communications, pages 1847-1851, 1996.

[13] M. H. Ammar and J. W. Wong, “On the Optimality of Cyclic Transmission in Teletext Systems”, IEEE Transaction on Communication, COM-35(1):68-73, January 1987.

[14] S. Acharya, M. Franklin, S. Zdonik, “Dissemination-based Data Delivery Using Broadcast Disks”, IEEE Personal Communications, 2(6), 1995.

[15] J. W. Wong, “Broadcast Delivery”, in Proc. of IEEE, pp. 1566-1577, December 1988.

[16] M. H. Ammar, “Response Time ina a Teletext System: an Individual User’s Perspective”, IEEE Trans. On Communications, COM-35(11):1159-1170, November 1987.

[17] R. Alonso, D. Barbara, H. Garcia-Molina, “Data Caching Issues in an Information Retrieval System”, TODS 15(3):359-384(1990).

[18] B.L. Worthington, G.R. Ganger, and Y. Patt, “Scheduling algorithms for modern disk drivers”, in Proc. of the 1994 ACM SIGMETRICS Conf., pages 241-251.